HPF/ES プログラミング

― 実際にあった問題とその解決

村井 均

地球シミュレータセンター

murai@jamstec.go.jp

はじめに

- 発表者は、地球シミュレータセンター(ESC)において、プログラムサポートの業務を行っている。
- HPF/ESユーザが出会う、
  - よくある問題
  - その解決法

  を紹介する。

- 性能強化と機能強化に関する要望を述べる。

もくじ

- 並列ループ中のCALL
- 外来手続き
- 入出力
- 階層並列化
- 周期境界条件
- 配列の転置
- パイプライン並列化

並列ループ内のCALL（1）

手続き呼び出しを含むループを並列化できる条件:

- ループにINDEPENDENTが指定されている。
- どの引数も分散されていない（配列要素や部分はOK）。
- 手続きの外来種別が、HPF_GLOBALかEXTRAN_LOCALである。

HPF_LOCALはNG
以前はできたはず…

…インタフェースブロックで宣言する。ただし、外来種別の既定値はHPF_GLOBALなので省略しても良い。

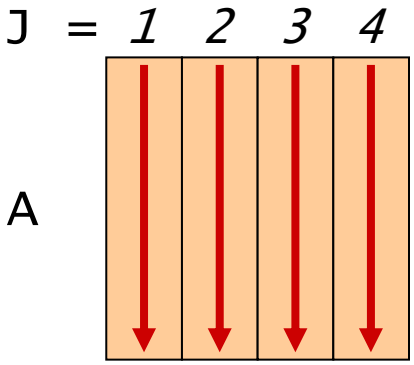
HPFまたはHPF/ESにおけるPUREの意味は？

【Note】手続きがPUREでないとワーニングが出る上に、ループにON HOMEを指定できない。

並列ループ内のCALL (2)

使用例

```
      REAL A(NX,NY)
!HPF$ DISTRIBUTE A(*,BLOCK)

!HPF$ INDEPENDENT
      DO J = 1, NY
        CALL SUB(A(:,J))
      END DO
```



- 現在は、必ずメモリコピ
- インライン展開を指定で
  もある。
- メモリコピーを伴わない

```
      REAL A(NX,NY)
!HPF$ DISTRIBUTE A(*,BLOCK)
      REAL T(NX)

!HPF$ INDEPENDENT
      DO J = 1, NY
        DO I = 1, NX
          T(I) = A(I,J)
        END DO
        CALL SUB(T)
      END DO
```

外来手続き（1）

- コンパイラが作る通信が遅い。
- Fortranで書かれたライブラリ(e.g. ASL)を使いたい。

➡ *外来手続きの利用*

外来種別	主な用途
HPF_LOCAL	MPIを使った通信や入出力の高速化
FORTRAN_LOCAL	Fortranで書かれたライブラリの呼び出し

外来手続き (2) HPF_LOCAL

```
      real A(N)
!HPF$ distribute (block) :: A
!HPF$ shadow (0:1) :: A
      ...
!HPFJ reflect A
```

```
      interface
        extrinsic(HPF_LOCAL)
     +  subroutine my_reflect(A)
        real A(:)
!HPF$   distribute (block) :: A
!HPF$   shadow (0:1) :: A
        end subroutine
      end interface
      ...
      call my_reflect(A)
```

```
      extrinsic(HPF_LOCAL)
     >   subroutine my_reflect(A)
      use HPF_LOCAL_LIBRARY
      include 'mpif.h'
      real A(:)
!HPF$ distribute (block) :: A
!HPF$ shadow (0:1) :: A
      comm = MPI_COMM_WORLD
      me = my_processor()
      np = number_of_processors()
      ...
      call MPI_Sendrecv(...)
      ...
      end
```

REFLECTが遅いので、MPIを使って高速化したい。

外来手続き (3) HPF_LOCAL

- 呼び側(caller)

  インタフェースブロック内で、MPI手続きを「HPF_LOCAL」として宣言する。

引数の分散次元には「:」を指定して、形状引継ぎ配列とする。

分散指定は、呼び出し前の状態を示す(必要に応じ、再分散が起こり得る)。

```
      interface
        extrinsic(HPF_LOCAL)
      +   subroutine my_reflect(A)
        real A(:)
!HPF$   distribute (block) :: A
!HPF$   shadow (0:1) :: A
        end subroutine
      end interface
      ...
      call my_reflect(A)
```

外来手続き (4) HPF_LOCAL

- 呼ばれる側(callee)
  HPF_LOCAL手続き
  として宣言する。

「mpif.h」をインクルードする。

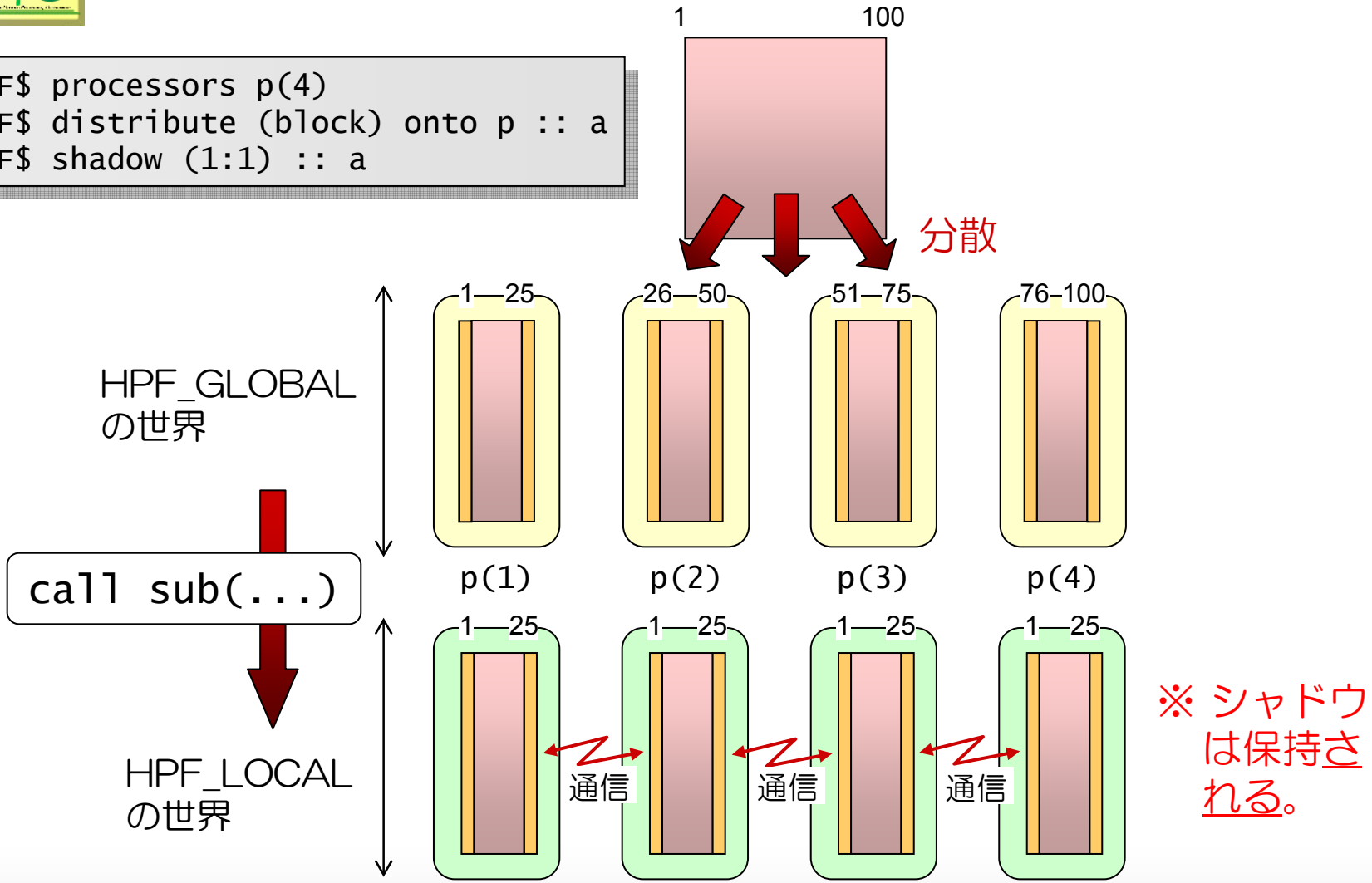
MPI_Initと
MPI_Finalizeを除く
任意のMPIライブラリ
を使える。

```
      extrinsic(HPF_LOCAL)
     >   subroutine my_reflect(A)
      use HPF_LOCAL_LIBRARY
      include 'mpif.h'
      real A(:)
!HPF$ distribute (block) :: A
!HPF$ shadow (0:1) :: A
      comm = MPI_COMM_WORLD
      np = number_of_processors()
      me = my_processor()
      ...
      call MPI_Sendrecv(...)
      ...
      end
```

※ 以下が保証される。
- コミュニケータ == MPI_COMM_WORLD
- MPIプロセス数 ==
  number_or_processors()
- MPIランク == my_processor()

外来手続き (5) HPF_LOCAL

```
!HPF$ processors p(4)
!HPF$ distribute (block) onto p :: a
!HPF$ shadow (1:1) :: a
```



外来手続き (6) FORTRAN_LOCAL

Fortranで書かれた既存のライブラリ(e.g. ASL)を使いたい。

➡ ASLサブルーチンを、FORTRAN_LOCAL手続きとして呼ぶ。

```
interface
  extrinsic(FORTRAN_LOCAL)
+ subroutine ASL_sub(...)
  ...
  end subroutine
end interface
...
call ASL_sub(...)
```

＋ ASLサブルーチン

外来手続き（7）FORTRAN_LOCAL

呼び側(HPFプログラム)

インタフェースブロック内で、ASLサブルーチンを「FORTRAN_LOCAL」として宣言する。

注意点

A) 引数として、ローカル配列のサイズを渡す。

B) 配列引数は、最終次元を分割する方が良い。

C) 最終次元以外を分割するときは、シャドウのサイズを0にする方が良い。

一次元FFTライブラリを呼び出す例

B)

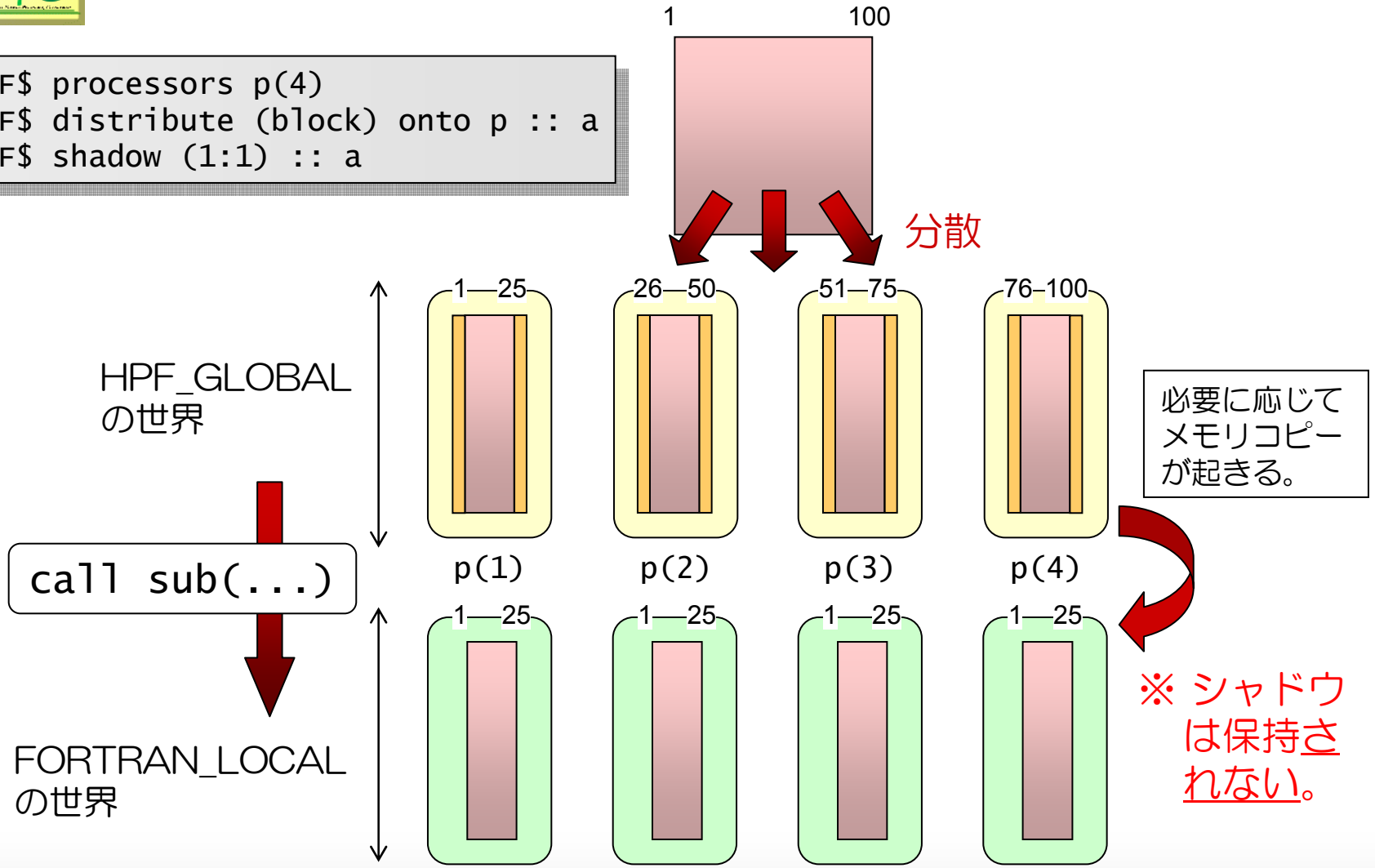
C)

A)

```
      real r(m,n), wk(m,n)
!HPF$ distribute (*,block) :: r, wk
!HPF$ shadow (0,0) :: r, wk

      interface
        extrinsic(FORTRAN_LOCAL)
     +  subroutine DFRMBF()
        real r(:,:), wk(:,:)
!HPF$   distribute (*,block) :: r, wk
!HPF$   shadow (0,0) :: r, wk
        ...
        end subroutine
      end interface

      np = number_of_processors()
      call DFRMBF(m,n/np,r,n/np,1,
     +      isw,ifax,trigs,wk,ierr)
```

外来手続き (8) FORTRAN_LOCAL

```
!HPF$ processors p(4)
!HPF$ distribute (block) onto p :: a
!HPF$ shadow (1:1) :: a
```



入出力（1）

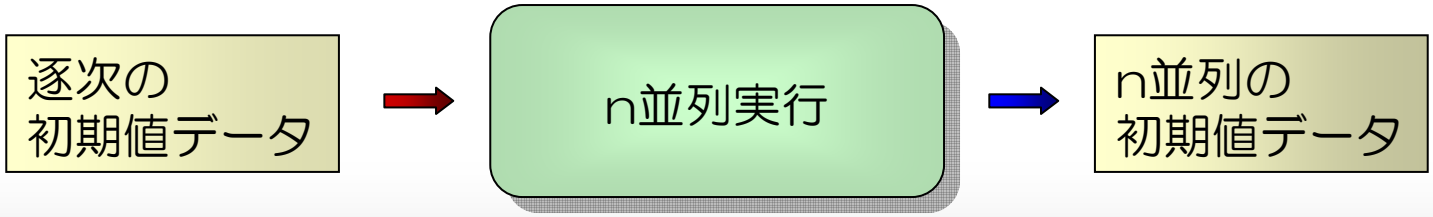
並列入出力

再分割ツールは、最初(初期値)と最後(計算結果)で必要になる。

- aligneeを扱えない → バージョンアップにより対応
- GEN_BLOCKはまだ扱えない。

最初と最後では、HPFプログラム内での再分割(並列ファイルへの変換)が必要になる。

```
!HPF$ DISTRIBUTE A(BLOCK)
!HPF$ DYNAMIC A
      READ(10) A
!HPF$ REDISTRIBUTE A(GEN_BLOCK(M))
      WRITE(11) A
```

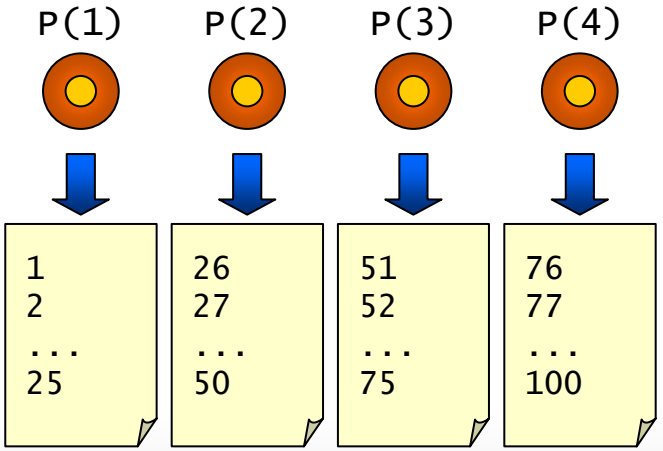


入出力 (2)

プロセッサ別の(書式付き)入出力

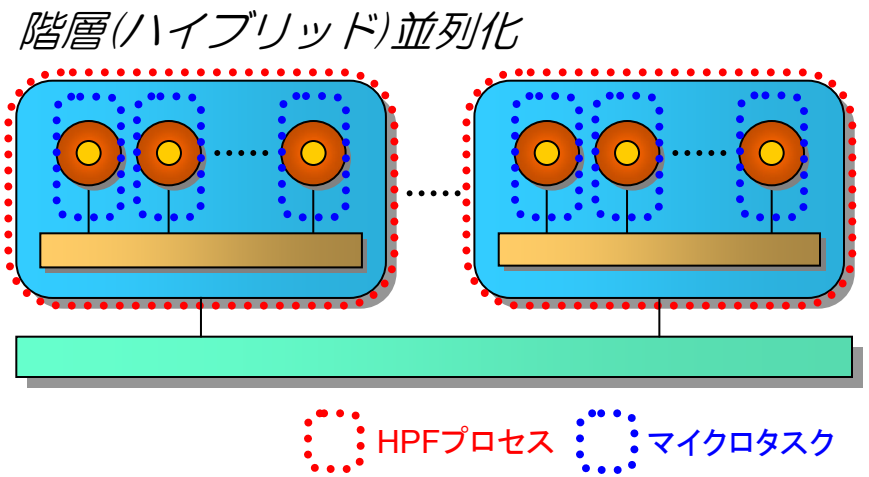
- 主にデバッグとコントロールデータの読み込みの場面で有用。
- 現在のHPF/ESでは不可。
- 中間Fortranソースを直接に編集すれば可。

```
!HPF$ INDEPENDENT
      DO I=1, N
        A(I) = ...
        WRITE(*,*) A(I)
      END DO
```



階層並列化（1）

FORTRAN90/ESの「自動並列化」機能とHPFを組み合わせて、階層並列化を実現できる。



利用可能な「自動並列化指示行」

- CONCUR|NOCONCUR
- SELECT
- SKIP
- SYNC|NOSYNC
- THRESHOLD|NOTHRESHOLD
- PARALLEL DO（新）

階層並列化 (2)

「自動並列化指示行」は、INDEPENDENTループ(を含む多重ループ)にしか指定できない。

次のようなループが自動的に並列化されない場合、通常は指示行を指定できない。

- 非分散配列を処理するループ
- 並列ループ中から呼ばれる手続き内のループ

```
!CDIR CONCUR
      DO I=1, N
        A(I) = ...
      END DO
```

Aは非分散

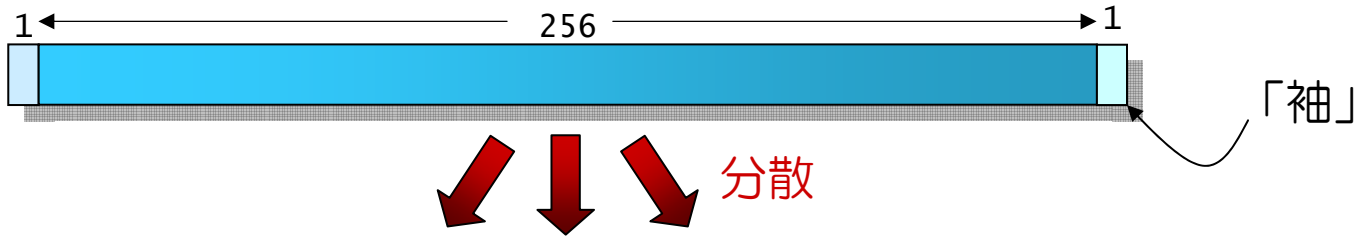
ダミーのINDEPENDENTループを追加。

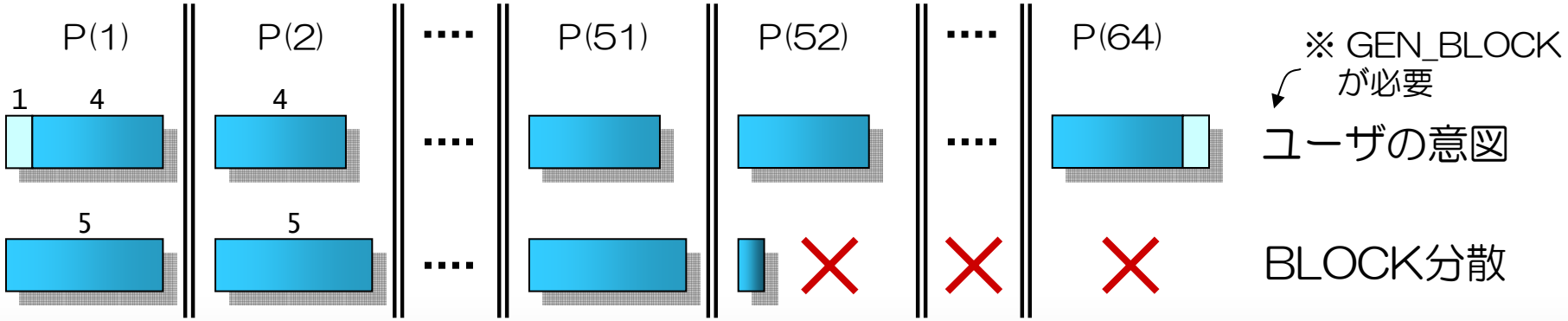
```
!HPF$ TEMPLATE T(NPROCS)
!HPF$ DISTRIBUTE T(BLOCK)

!HPF$ INDEPENDENT
      DO J=1, NPROCS
!HPF$   ON HOME(T(J))
!CDIR   CONCUR
        DO I=1, N
          A(I) = ...
        END DO
      END DO
```

周期境界条件（1）

```
      parameter (N=256)
      real a(N+2)
!HPF$ processors p(64)
!HPF$ distribute a(block) onto p
```

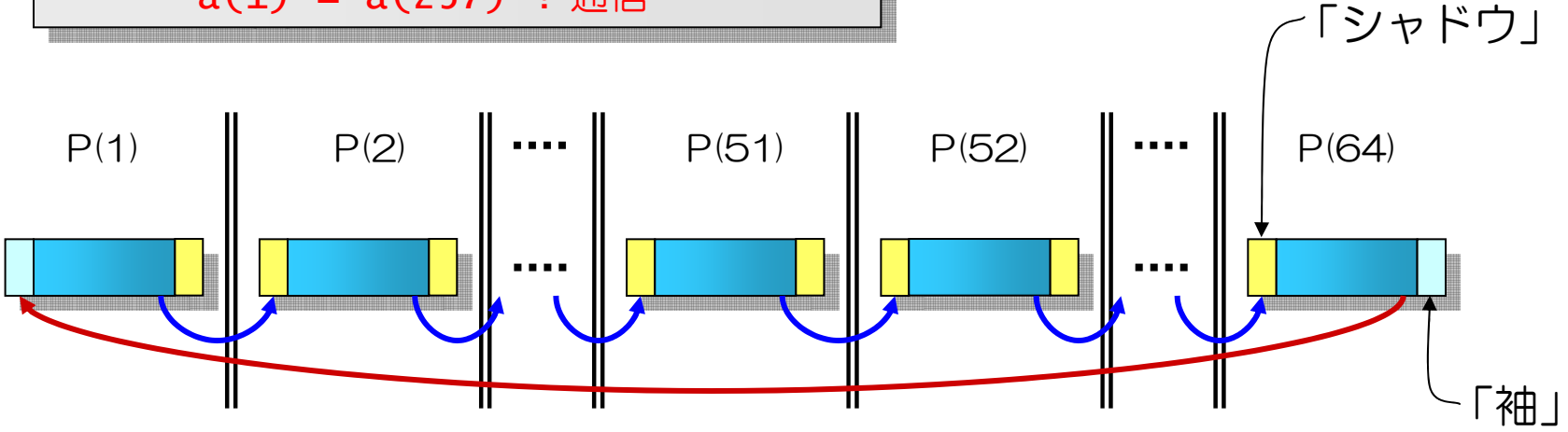
周期境界条件のために、元の配列が「袖」を持っていると、負荷の不均衡が起きる。





周期境界条件 (2)

```
!HPF$ shadow (1:1) :: a
!HPFJ reflect a      ! 通信
      a(1) = a(257) ! 通信
```



境界部の袖領域を更新する通信と、シャドウを更新する通信を同時に実行できると良い。

→ *cyclic shift*

配列の転置

代表的な性能ボトルネック

コンパイラが生成する通信とそれに伴うメモリコピーのオーバヘッドが大きい。

```
!HPF$ DISTRIBUTE A(*,BLOCK)
!HPF$ DISTRIBUTE B(BLOCK,*)
      B = A
```

```
      INTERFACE
        EXTRINSIC(HPF_LOCAL)
      >   SUBROUTINE SUB(A,B)
        ...
      END INTERFACE
      ...
      CALL SUB(A,B)
```

開発元(NEC)と協力し、特にメモリコピー部分のアルゴリズム改良により約30%の高速化。

ESのグローバルメモリと片側通信を利用すれば、さらに高速になることがわかっている。

※ MPI手続き(HPF_LOCAL)を利用する方法もある。

パイプライン並列化（1）

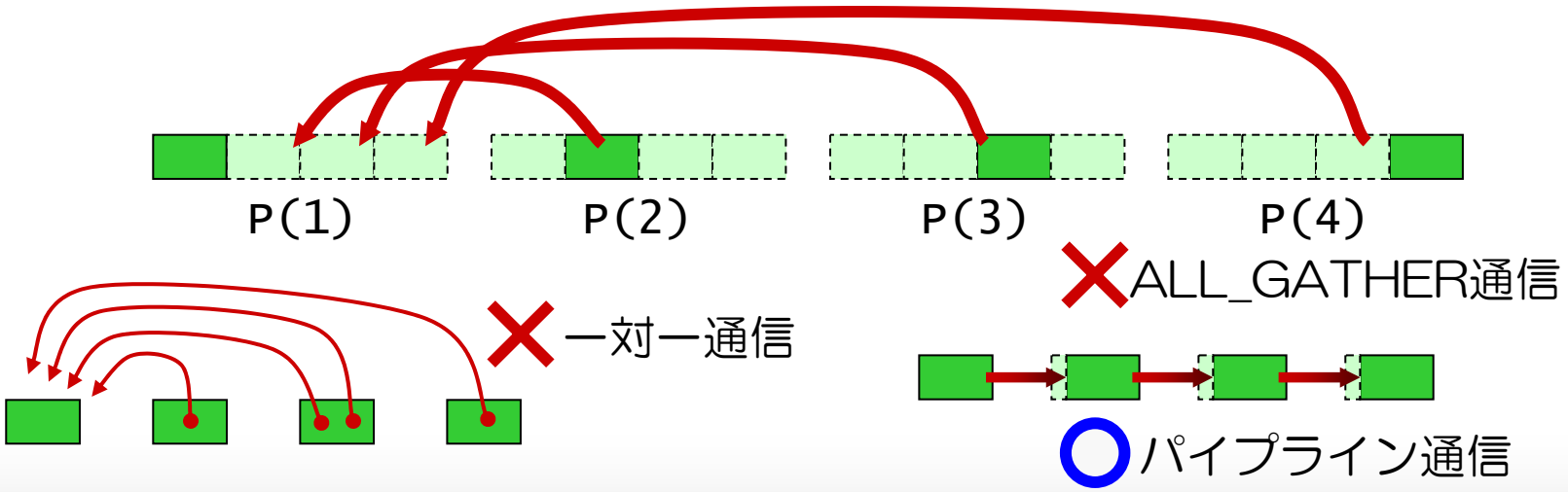
HPF/ESでは、DOACROSSループをうまく扱えない(重い通信が発生する)。

e.g. スキャッタ

```
DO I=1, N
  A(I) = B(IDX(I))
END DO
```

e.g. 累計

```
DO I=1, N-1
  A(I) = A(I) + A(I-1)
END DO
```



パイプライン並列化 (2)

簡単なパイプライン並列化を実現するプリプロセッサを開発。

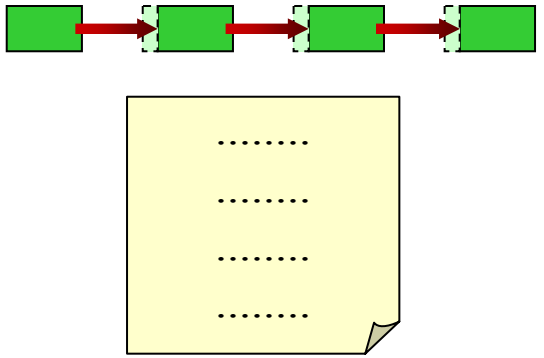
```
!HPFX PIPELINE(A(:),(1))
!HPF$ INDEPENDENT
      DO I=1, N-1
!HPF$   ON HOME(A(I)), LOCAL
        A(I) = A(I) + A(I-1)
      END DO
```

1. PIPELINE指示文で、対象の配列と通信の幅を指示

```
% hpfx -o test test.hpf
```

2. ドライバhpfxで、プリプロセッサの起動, コンパイル, リンクを実行

3. パイプライン並列化を実現したオブジェクトを生成



パイプライン並列化 (3)

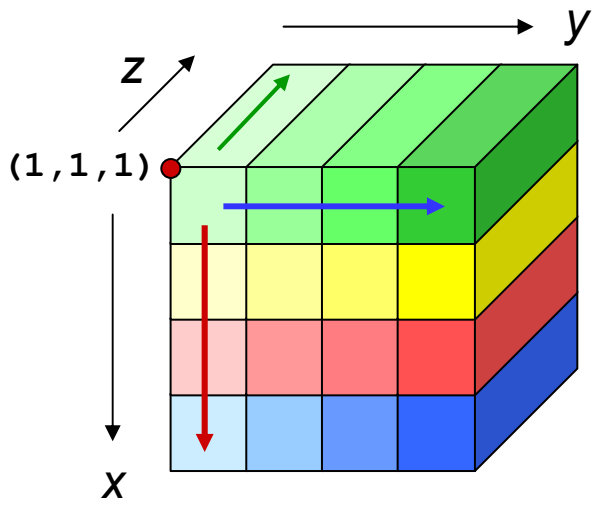
NPB LUでの評価

- 依存関係から、単純なDOALL並列化は不可
- 従来の簡易ハイパープレーンはオーバヘッドが大きい。

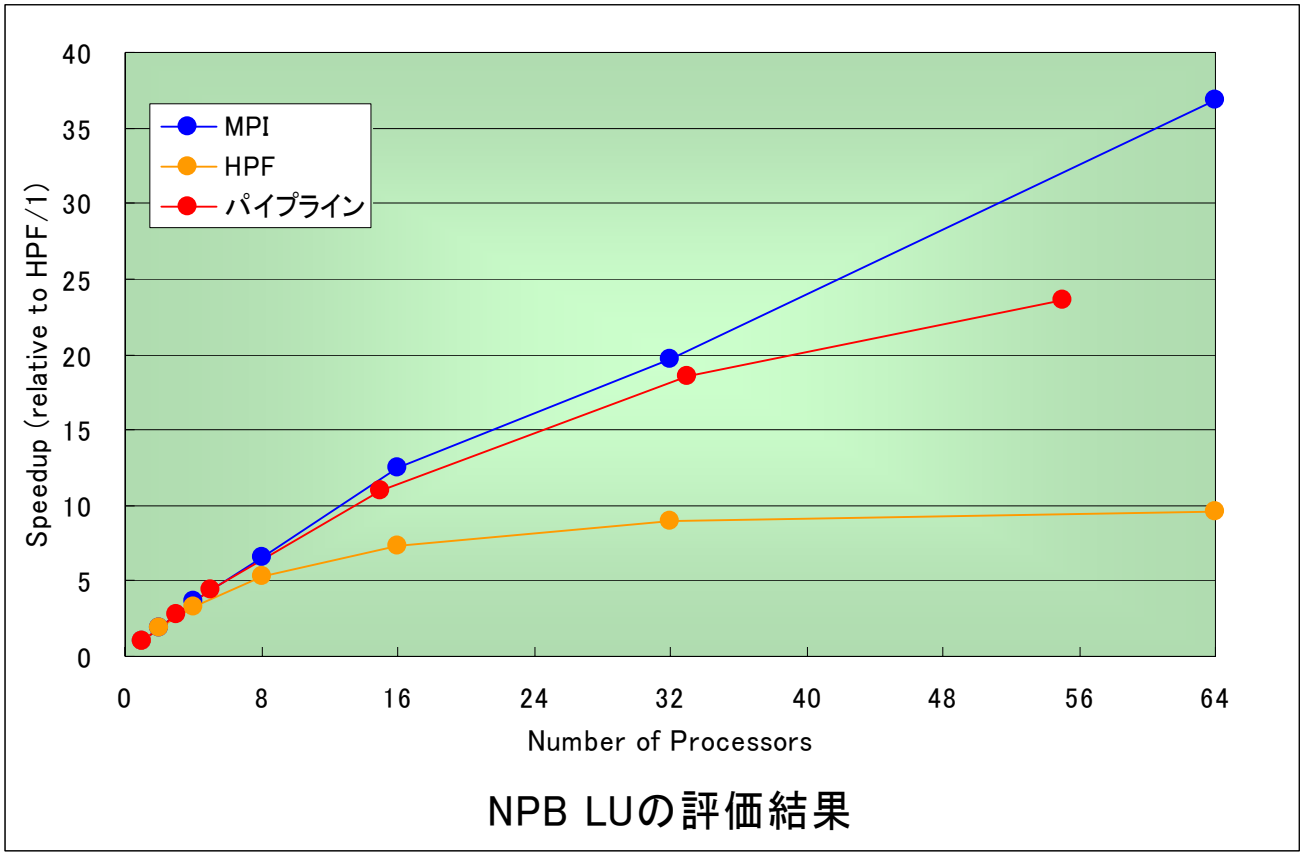
➡ *開発したプリプロセッサによりパイプライン並列化*

データの依存関係:

```
a(i,j,k) ← a(i-1,j-1,k-1)
```



パイプライン並列化 (4)



NPB LUの評価結果

- 地球シミュレータのL系(大規模)バッチジョブで評価
- フラット並列化
- クラスC
- 逐次版(NPB2.3-serial)をHPFで並列化
- MPI版は、NPB2.4のコードを使用

まとめ

性能強化と機能強化の要望

- 並列ループ内CALLの高速化(メモリコピーの削除)
- HPFまたはHPF/ESにおけるPUREの意味
- 逐次ループに対する自動並列化指示行
- 周期境界条件
- 配列転置の高速化
- パイプライン並列化